LA-UR-19-20745
Approved for public release; distribution is unlimited.

Title: Continuous Integration! You Keep Using Those Words. I Do Not Think They Mean What You Think They Mean.

Author(s): Adams, Terry R.

Intended for: Report

Issued: 2019-09-12 (rev.3)

Continuous Integration!

You keep using those words.
I do not think they mean what you think they mean.

Terry R. Adams

Monte Carlo Methods, Codes, and Applications Group (XCP-3); X-Computational Physics Division
P.O. Box 1663, Los Alamos National Laboratory, Los Alamos, New Mexico 87545 USA
26 February 2019, version 4.0
LA-UR-19-20745

Abstract

Continuous Integration (CI) is a best practice in the software community. Often though, there are several ideas as to what it means to practice continuous integration. With this uncertainty of the definition often comes a loss of benefit. It is good to stop and reflect on how and why CI has been, and still is, practiced in software community. So, presented here is a historical prospective of CI along with associated practices, anti-patterns, and benefits. A subsequent presentation will discuss the workflows (including code reviews) and the controversy that arises as CI has changed with time, especially in regards to feature-branch development.

Keywords: Introduction, Continuous Integration, Software Development, MCATK

Email address: drbubba@lanl.gov (Terry R. Adams)

1. Introduction

This is an introduction to an established practice of modern software development, called *continuous integration* (CI). The purpose of this document is to supplement an oral presentation about CI (Adams, 2019), and to provide a resource to help the scientific software development (SSD) community, especially at Los Alamos National Laboratory.

I have observed, in the literature and in conversations with other developers, that the perceived meaning of CI has evolved and is perhaps now more diffuse compared to the 'historical' definition that I trained with and learned. Fowler (2006b) uses an expression for this type of change, which he calls "semantic diffusion". He points out that as these changes happen they often "spread through the wider community in a way that weakens that definition. This weakening risks losing the definition entirely - and with it any usefulness to the term."

A generic refrain often heard, when CI is brought up in a conversation with other developers, goes as follows: "We do continuous integration. We use [insert your favorite CI software tool name]." And often they add, "and it automatically runs builds and tests when we make a commit (or a pull request)." These type of conversations leave me perplexed and make me wonder if they know what they are missing. So, I decided to create a presentation and this paper to remind scientific developers of the history of integration, specifically continuous integration. This decision was recently reinforced by a Kevlin Henney presentation (Henney, 2018) and comment, "Something else in software that we are particularly bad at, we have a weak sense of history".

Continuous integration is a practice, not a software tool; CI is an attitude, as much as it is a practice; CI is as much about what happens before a commit, as to what happens during and after a commit.

In software development, the integration of components has been an essential step in the process almost since first programming a computer. Every project has to deal, and often struggle, with the integration process. With time, several practices have evolved for dealing with this important software development step. With a focus on leading to CI, the following list highlights a select few of the advances in the practice of software integration:

- (1980s) Microsoft famed practice of "Daily Builds". (Cusumano and Selby, 1995).
- (1991) First published use of the term "Continuous Integration". Integration made possible in the context of OO[1] Design. (Booch, 1991)
- (1998) "Integrate and test several times a day". First publication of Extreme Programming (XP). (Beck, 1998).
- (1999) Embracing Change with XP paper. Beck's first publication with the term CI. CI included as one of several practices for XP. (Beck, 1999).
- (2000) Extreme Programming book (Seminal). A CI Description (Beck, 2000).
- (2000) Continuous Integration paper (Classic). Established a CI outline and practices (Fowler and Foemmel, 2000, 2006)
- (2007) Continuous Integration book (Definitive). (Duvall et al., 2007)
- (2010) Continuous Delivery book (Definitive). Continuous delivery is built upon CI. (Humble and Farley, 2010)

In the following sections, we will discuss a few of these advances with the focus being on the practice of continuous integration. A bit of background is given in Section 2 about software integration up to the point that CI arrives on the software development scene. Section 3 is about a historical description of CI with discussions on some practices, benefits, and antipatterns of CI. Brief discussions about two somewhat contentious CI practices, daily integration and trunk-based development are found in Sections 4 and 5, respectively. In Appendix A is the basic CI description of the Monte Carlo Application ToolKit (MCATK) (Adams et al., 2015) project. Finally, in the Bibliography at the end of the document (along with the work cited in this paper) is gathered the full bibliography of the author's research sources on this topic.

[1] OO stands for object-oriented.

2. Integration

> "*Integrating* a software project means taking the components of the software as written so far, compiling them together and running the tests (the regression suite)."
> – Bertrand Meyer (Meyer, 2014a, p. 103)

This definition is concise and self-explanatory. Yet, it does not address the "amount of change" and "interval duration" with which the integration happens on a project. These two points are at the heart of modern software integration, especially CI.

2.1. Integration Approaches: Phased and Incremental

McConnell (2004) writes about two software integration approaches described as phased and incremental. *Phased integration* follows several defined phases. Phase one, is to design, code, and test each component. The second phase, is to take the components and combine them into one large system. And the last phase is to test (and debug) the whole system. Since this approach is bringing together (after an extended period of time) components that have not been integrated or tested before, then this only compounds the integration process and makes it harder to locate the problems during integration and testing. For this reason, a more dramatic name for phased integration, as mentioned by McConnell (2004), is *big bang integration* (Booch, 1994; Meyer, 2014a). The term *integration hell* is often the resultant state of a "big bang" approach to integration (Cunningham and Contributors, Last Edited 2012)(Duvall et al., 2007, p. xix). Of course, one does not have to follow these defined steps to have big bang integration. Just by having *many changes* and *longer times* in between integrations can often bring about integration hell.

The second integration approach that McConnell (2004) describes is called *incremental integration*[2] (Booch, 1991, 1994, 1996; Cockburn, 2005). Incremental integration is, just as it says, in increments. The increments can be small pieces of functionality in either classes or components that are developed and tested alone but then these pieces are gradually integrated and tested into the full system sooner than later. This way the system integration and testing is built up incrementally letting each developer see the changes to the system from other developers on the team sooner. Thus, it is helping to spread the "pain" of integration and testing out over the duration of the project rather than dealing with it all at the end in a "big bang" event.

Several benefits listed by McConnell (2004, p. 693) for incremental integration are: errors are easier to locate, the system succeeds early in the project, you get improved progress monitoring, you will improve customer relations, the units of the system are tested more fully, and you can build the system on a shorter development schedule.

Incremental integration provides faster feedback

2.2. The "Daily Build"

As listed in the Introduction (Section 1), one of the more visible advances in integration practices came in the 1980s and might be considered an incremental integration approach.[3] At that time, Microsoft developed the practice of the *Daily Build* (DB) which is described in interesting detail by Cusumano and Selby (1995, p. 263).

The DB is the practice of stopping all source commits to the main repository at the end of the work day, running a build of the integrated source, and executing basic tests that covers a judicious, yet significant, set of end-to-end tests to provide quick[4] feedback about the present state of the integrated mainline repository.

Cusumano and Selby (1995, p. 267) states developers usually integrated their work at least twice a week though they could do so daily. By spreading the "pain" of integration over the project duration instead of all near the end, lead a software engineering manager for Windows NT to say, "Doing the daily builds is just like the most painful thing in the world. But it is the greatest thing in the world, because you get instant feedback." (Cusumano and Selby, 1995, p. 268)

[2] The idea of incremental, iterative, and evolutionary software methods have been around since the 1960s. See Gilb (1981, 1988), Larman (2004, p. 9, 79) and the references within.

[3] And, if not thought of as an incremental approach, then at least can be used in conjunction with incremental approaches.

[4] An example of run-times for what they call "quick tests" is taken from the Excel project which took about 30 minutes to execute.

According to McConnell (2004), some of the benefits of DB are: it minimizes integration risk, it reduces the risk of low quality, it supports easier defect diagnosis, and it improves morale. Also, like incremental integration, it helps spread the "pain" of integration and testing out over the duration of the project.

There are several things to note about the DB practice which are: the daily synchronization cadence of integrated code, a suite of "quick tests" which are available for use by the developers all during the development phase, issues with the build & tests are expected to be fixed immediately, faster feedback, and the integration to a centralized mainline code repository.

Beside the excellent description of DB by Cusumano and Selby (1995), which I highly recommend, this topic is also discussed in Steve McConnell's two quintessential software engineering books called Rapid Development and Code Complete (McConnell, 1996b, 2004). If the reader does not have easy access to these books then perhaps one can find the nice condensed version called "Daily Build and Smoke Test"[5] provided by McConnell (1996a).

Many might consider continuous integration to rest on the shoulders of the daily build process and to be a next step in the evolution of integration approaches. Several in the software industry, especially agile practitioners, consider the DB approach as a minimum needed (Fowler and Foemmel, 2000; Shore and Warden, 2008).

3. Continuous Integration

> "*Continuous Integration*[6] is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily – leading to multiple integrations per day. Each integration is verified by a 'build' (including tests) to detect integration errors as quickly as possible." – Martin Fowler (Fowler and Foemmel, 2000)

In Booch's book entitled, Objected-Oriented Design: With Applications, he states, "A far better way to measure productivity derives from the practice of continuous integration. In this evolutionary approach, there is no big-bang integration event;" (Booch, 1991, p 209). From what the author can discern, this is the first documented uses of the term *continuous integration* and is used in all three of Booch's books from the 1990s which are in the context of objected-oriented programming, integration, and design; and describes an incremental and iterative process (Booch, 1991, 1994, 1996).

In the 1990s, Kent Beck included CI as one of the 12 practices within the Extreme Programming (XP) software methodology (Beck, 1998, 1999, 2000). He was very specific with his description for CI, "No code sits unintegrated for more than a couple of hours. At the end of every development episode, the code is integrated with the latest release and all the tests must run at 100%" (Beck, 2000, p. 97). In other words, developers integrate very often to the main repository with *NO* build or test failures. These seminal publications on XP, with the inclusion of CI, helped to start quantifying CI as a software practice.

In a classic web-based article (Fowler and Foemmel, 2000), Martin Fowler described how the developers at ThoughtWorks were setting up and practicing CI. Fowler sets the tone of the paper by saying, "We are using the term Continuous Integration, a term used as one of the practices of XP (Extreme Programming). However we recognize that the practice has

[5] The Cusumano DB description includes use of "quick tests", and mentions they are often referred to as "smoke tests" (Cusumano and Selby, 1995, p.265).

[6] Since there are intervals, and not a true continuous flow in CI, some suggest that *continual integration* might be a better name than continuous integration (Cunningham and Contributors, Last Edited 2014; Warden, 2003; Duvall et al., 2007).

been around for a long time and is used by plenty of folks that would never consider XP for their work." Though, Beck and Fowler were not the creators of CI, they were influential in steering the evolution of software integration from then until the present.

3.1. Continuous Integration Practices

As is seen in the Fowler's second edition of this paper (Fowler and Foemmel, 2006), there is a list of suggested practices within CI that are still the basis of many CI implementations and are often referred to by others. In this list the word "Build" means to build and test. The list of CI practices are:

Continuous Integration Practices

- Maintain a Single Source Repository [Mainline]
- Automate the Build
- Make Your Build Self-Testing
- Everyone Commits to the Mainline Every Day
- Every Commit Should Build the Mainline on an Integration Machine
- Fix Broken Builds Immediately
- Keep the Build Fast
- Test in a Clone of the Production Environment
- Make It Easy for Anyone to Get the Latest Executable
- Everyone Can See What is Happening
- Automate Deployment

Projects do not have to start with all of these practices at once, but can select a few at a time and build on that with some of the others practices. If one looks more closely at several of these practices then it should be clear that for developers there needs to be an proactive attitude with as much going on before the commit as during and after. With CI, as with the basic trends described in earlier sections, it is good to keep development in small increments and iterations. This is reflected and reinforced in the practices of self-testing Builds, daily commits by everyone, fast Builds, immediately fixing broken Builds, and everyone seeing what is happening. Of the listed practices, only few will be touched on here. For more details and ideas about CI, see Fowler's papers (Fowler and Foemmel, 2000, 2006), the definitive book by Duvall et al. (2007), and the CI chapter in Humble and Farley (2010).

When it says to "Keep the Build Fast", one has to ask, "How fast is fast?" A number of folks have suggested that a *Ten-Minute Build* (TMB) is reasonable for most projects (Beck and Andres, 2004; Fowler and Foemmel, 2006; Duvall et al., 2007; Shore and Warden, 2008). The TMB rule means building and testing in ten minutes. Your mileage may vary. Similar to the DB (see Section 2.2) the tests should be quick tests which now include unit tests.

The practice of "Everyone Commits to the Mainline Every Day" is the most controversial of the CI practices. Controversial less for the "daily commit" frequency (see Section 4) and more for the "committing to the Mainline" part of the practice. Due to this, a discussion about Trunk-Based development will be expanded a bit in Section 5 and more so in a future presentation that focuses more on the topic.

Given that, we focus in this section on one of the reasons for "Everyone Commits to the Mainline Every Day" which is that of communication. Daily, mainline integration allows developers to communicate their changes to each other in a timely and visible manner. Hence, the point of the word integration in the term continuous integration.

It is often stated, one of the goals of CI is frequent deployment and every successful Build should be a candidate for release (Fowler and Foemmel, 2006; Shore and Warden, 2008; Duvall et al., 2007). Thus, the last CI practice listed is "automate deployment." Also, this is why many consider CI as the backbone of *continuous delivery* (CD). CD takes the next step and is the practice of setting up a "pipeline" in the development process that takes the product from developing code, to integrating the code, to preparing it for release, and releasing it frequently. Again, frequent means with every repository commit. All of this, hopefully, transparent to the users. For more details about CD, see the definitive book by Humble and Farley (2010).

I will add one more practice, which is implicit in this list, that is "Run a Private Build". Once your development "sandbox" is building and executing the

tests successfully then merge into your development "sandbox" any changes from the mainline repository. Then execute a clean, new Build. With no failures, you are ready to merge your changes to the mainline repository.

As stated in the Introduction (Section 1), CI is a practice and not a tool; Its an attitude[7] as much as a practice. This means that to adopt its use requires an explicit commitment from the development team before moving forward.

I list here some of the the benefits of CI, but not all:

Continuous Integration Benefits

- Reduces risks to integration and the overall project
- Increases visibility and communication of code changes between developers
- Establishes greater confidence in the software product for developers, managers, and users
- Increases product visibility for decisions and trends
- Generates deployable software at any time, any place, and frequently
- Reduces repetitive manual processes
- Helps the human factor with natural breaks, increased confidence, and morale

We will leave this list as it is and refer the reader to three references (Beck, 2000; Fowler and Foemmel, 2006; Duvall et al., 2007) for more details and benefits.

Note that CI improves on the two points called out in the Integration section (Section 2) on the "amount of change" and "interval duration". Also, with the Incremental and DB integration legacy, CI is built for (and improves on) fast feedback and reducing risks to the project.

Integrate Often
Keep Batch Size Small
Be Prepared to Deploy

[7] See Shore (2005a) web post about attitude.

3.2. *Continuous Integration Anti-Patterns*

An *anti-pattern* (AP) is an intentional or unintentional practice (or habit) that negates (or works against) one or more practices in a process. Duvall (2007, 2008) has written about anti-patterns of CI[8] which are gathered here:

CI Anti-Pattern Names

1. Infrequent Check-Ins
2. Broken Builds
3. Minimal Feedback
4. Receiving Spam Feedback
5. Slow Builds
6. Bloated Build
7. Bottleneck Commits
8. Continuous Ignorance
9. Scheduled Builds
10. Works On My Machine
11. IDE-Only Build
12. Myopic Environment
13. Polluted Environment

The first six are mostly self-explanatory and are the primary APs that the other ones are related to or built upon. Lets examine several of the secondary APs that are less obvious. The "Bottleneck Commits" are caused by developers committing code changes at the end of the day just before going home. This AP is a variation on the "Infrequent Check-Ins" anti-pattern. Another AP variation is called "Five-O'Clock Checkin" which should speaks for itself.

"Continuous Ignorance" is when successful Builds, one after the other, are masking the fact that defects are making their way into the integrated system. This is often due to the Build being made up of simple compilation and a few unit tests. The team is lulled into a false sense of security. By adding additional processes like more focused integrated tests, automated code inspections, and so on, you are better able

[8] Also see Shore (2008) for forces working against CI.

to determine if the software is actually working earlier in the development cycle. But remember that the more you add, the slower the feedback becomes.

"Scheduled Builds" is basically Builds that run daily, weekly, or on some other schedule, but *NOT* with every commit. Remember, if Builds are run on a less frequent basis than with every commit then you are delaying the discovery of issues earlier in the development cycle.

"Polluted Environment" is where a partial Build is ran to save time. This can lead to false positives or false negatives. It is better to clean previously built artifacts prior to running a Build. Some like to run a *a scorched earth* policy (Duvall, 2008) where you start everything from scratch so to speak.

4. Continuous Integration: Daily Integration!?

We have shown in this report that CI has been influenced by Agile with its 'commit, build, and test on a daily basis, if not several times a day' practice. Some software projects are not as comfortable with this approach and might wonder what would be a good cadence for any given project.

One of the main goals in CI is to *always* be prepared to deploy the software at a moments notice. With a comment that supports this goal, the Poppendieck's wrote:

> "It is not always practical to integrate all of the code all the time. How often you integrate and test depends on what it takes to find defects... The proof that you are integrating frequently enough lies in your ability to integrate rapidly at any time, without finding defects." – Mary and Tom Poppendieck (Poppendieck and Poppendieck, 2010, p.78)

So, the reader might want to ask themselves,"What integration cadence[9] is needed for my project to support this statement?" or "How often do we find defects when we integrate, and would shortening the integration cadence help?"

Though the Poppendieck's give practical advice for an integration cadence, the next section highlights research that indicates a daily cadence should still be strongly considered.

[9] See Poppendieck and Poppendieck (2010, p.78) for thoughts on cadence.

5. Continuous Integration: Trunk-Based Development

One aspect of continuous integration that seems to be the most controversial in the software community is *trunk-based development* (TBD)[10]. An illustration of the controversy[11] can be seen in a recent blog post by Farley[12] and the unusually large number of responses in the comments (Farley, 2018c).

When starting the research on this report, I was unaware of this controversy in CI; though I did perceive[13] an underlying tension between TBD and *feature-branch development* (FBD)[14], which was one of the reasons I started this research on CI. Also, coworkers were saying they were doing CI even though they were using long lived branches and the GitHub-like pull-request approach which seemingly slows down the frequency of integration. It all seemed contradictory to someone, like me, who had been doing CI in a TBD approach. I was concerned that no one else felt this tension between these two, almost seemingly, orthogonal approaches.

Turns out, I was not alone and there happened to be a good bit of discussion on this topic in the software community. A particularly useful presentation by Sam Newman, at the GOTO Conference, called "Feature Branches and Toggles in a Post-GitHub World" (Newman, 2017) and explains a good bit of integration history and the controversy on this topic. Though Newman is biased toward TBD, he gives a fairly balanced presentation on the topic. He explains CI and TBD and continues on to illustrate the rise and use of GitHub and pull-based requests.

Focus here will remain on TBD and save the broader discussion for TBD versus FBD for a later date. So following up on TBD, there has been a series of studies that included the very topics of CI and TBD. Researchers at DevOps Research & Assessment (DORA) have been releasing annual reports since 2014 (Forsgren et al., 2014, 2015, 2016, 2017, 2018a) and have published a supporting book that gives explanations, reasonings, and details that the reports do not (Forsgren et al., 2018b). The results are based on surveys from the software community. And though the study was started for DevOps[15] it covers a broad cross section of the software community.

Their research "uncovered 24 key capabilities that drive improvements in software delivery performance" (Forsgren et al., 2018b, p. xix). Of those capabilities, two are the most relevant to this report. They are CI and TBD. Note that DORA's CI definition (Forsgren et al., 2018b, p. 44) is consistent with this reports description of CI (see Section 3). Their results related to CI and TBD show the following practices contribute to higher software delivery performance (Forsgren et al., 2016, p.30-31):

- Merging Code into Trunk on a Daily Basis
- Having Branches that are Short Lived (Less than a Day)
- Having Fewer than Three Active Branches

Although DORA's findings in the 2016 report showed "abundant evidence" (Forsgren et al., 2017, p.40) that TBD practices contributes to higher performing teams, there are developers who were [16] and remain skeptical, especially those familiar with a GitHub workflow (Forsgren et al., 2018b, p. 55). So DORA followed up in a 2017 study specifically looking at development workflow practices to see if there was differences between high, medium, and low performers. DORA reported in 2017 that the high performers used the shortest integration times and branch lifetimes. The low performers used the

[10] Trunk-based development means to integrate directly into the mainline repository.

[11] Also see https://www.reddit.com/r/programming/comments/7eko0m/trunk_based_development/

[12] Farley had been unofficially doing CI many years before XP and the rise of CI. He is a proponent of TBD plus speaks and writes on it often. He is also the co-author of the Continuous Delivery book.

[13] In part, to the MCATK project recently starting to use Gitflow.

[14] FBD uses a "GitHub Flow"-like workflow for development. This relies on developing on branches and merging back to the trunk, or mainline, repository periodically

[15] See Garnichaud (2012) for a description of DevOps

[16] See the DORA case study of Capital One for one of its senior director's, who was also leading the Continuous Delivery and DevOps transformation, comments of early skepticism and later significant improvements due in part to TBD (DORA, 2017).

longest integration times and branch lifetimes. And "these differences are statistically significant" (Forsgren et al., 2017, p. 41).

One proponent of TBD has written on the subject often and even has a very detailed website called TruckBasedDevelopment.com. The site goes into detail about integration and development, and I highly recommend it for more details about the pros & cons about this subject (Hammant and Smith, 2018).

It is important to say that there is nothing innately evil about FBD. Teams who are having success with it might consider TBD as an opportunity to improve based on the DORA results. Many feel there is nothing that makes FBD and CI incompatible as long you merge to the trunk daily, keep branches short lived (less than a day), and have fewer than three active branches. This would be consistent with what has been presented in this report for CI.

This section is brief since I plan to expand on this topic in a subsequent presentation. The brevity (most likely) created more questions than answers, so the bibliography includes a number of references on the topics of TBD and FBD, please refer to them for more on this topic. Also, the future report will explore modern code review which is conspicuously absent in this presentation.

6. Continuous Integration Users

A quick scan of the internet finds the following places use continuous integration as part of their software development. There are several references in the attached bibliography.

- IBM
- Microsoft
- Google
- Facebook
- Amazon
- Etsy
- Target
- Walmart
- Nordstrom
- Fidelity
- CapitalOne
- Intel
- Kitware
- Flickr
- LinkedIn
- Netflix
- Tesla
- Adobe
- Airbnb

It is important to point out that CI and TBD are the norm at several, if not more, of these organizations such as Google (Potvin and Levenberg, 2016) and Facebook (Feitelson et al., 2013; Rossi et al., 2016) to name a few. And note that code review is required and an important part of the workflow at many of these places (e.g., Google).

7. Summary

> ***I have told you the truth,***
> ***and it's up to you to live with it.***
> – inspired by Yeste of The Princess Bride

Continuous Integration is considered a best practice in software development. Interestingly enough, out of all the XP practices, Bertrand Meyer[17] considers Test-First and CI as the most lasting technical contributions to the software development industry and community (Meyer, 2014a, p.138). Meyer, also, includes CI under the Brilliant category of his Ugly, Hyped, and Good categories of Agile (Meyer, 2014a, p.154).

Continuous Integration is summarized in this report as integrate often, keep batch size small, and be prepared to deploy.

With all this discussion about the integration process, a *Challenge* is made to the reader: *Analyze your process, find ways to improve it, and in doing so strongly consider adding Continuous Integration ideas and practices.*

8. Acknowledgements

A special thanks to the other, initial MCATK, members (Tim Goorley, Jeremy Sweezy, Steve Nolen, and Tony Zukaitis) who willingly supported, and committed to, the practices of CI from the beginning of the project. Thanks to David Moulton, the T-5 DGL and Lead/POC of IDEAS, for funding support of these presentations. A thank you to Martin Fowler for kindly pointing me to the annual DORA "State of DevOps" reports and the more detailed accompanying book, Accelerate. And a nod to the late William Goldman for inspiration from his The Princess Bride novel.

[17] Meyer is a software engineer who invented Design by Contract and was one of the earliest proponents of object-oriented programming. See https://en.wikipedia.org/wiki/Bertrand_Meyer.

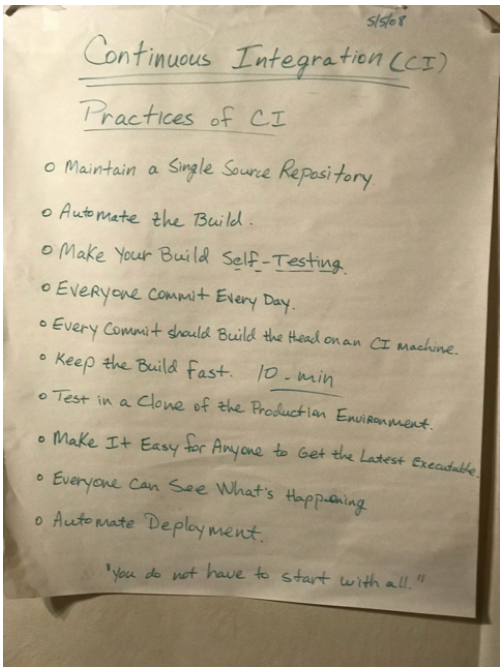

Figure A.1: CI Practices on a large Post-it note, dated 2008, which can be found in the MCATK Bullpen

Appendix A. MCATK Continuous Integration

The Monte Carlo Application ToolKit (MCATK) (Adams et al., 2015; Trahan et al., 2018) team practiced CI from the inception of the project in 2008. The team made a commitment and started with, and added most of, the basic CI Practices (see Section 3) which is posted on the MCATK bullpen wall, see Fig. A.1. The listed practices were used regularly as part of our CI process.

In the spirit of the simple CI process described by Shore (2006) in "Continuous Integration on a Dollar a Day", the following is the initial CI process MCATK started with and used until 2017. This process outline was taken from documentation in the MCATK TeamForge account dated July 2008.[18]

Please keep in mind as your read the following process outline, that it does not reflect that the team also practiced Self-Testing Builds, Run A Private Build, Ten-Minute Builds[19], working in small chunk & duration Commits Every Day (if not several times a day) and upon commit a clean, new checkout and Build was done on a Clone Production Environment.

MCATK Process For Continuous Integration

1. Build & Test Local "Sandbox"
   (a) make clean
   (b) make all/runtests (multi-compilers)
   (c) 100% NO errors (or warnings)
2. Update Local "Sandbox" With Head
   (a) svn update
   (b) Repeat Step 1).
3. Communicate Intention Of integration
   (a) Verbally and/or with a CI Token (Chucklehead the Sockmonkey)
   (b) Brief code "freeze"
4. Commit Your Code
   - svn ci -m "Really well tested and communicative code"
5. Go To CI Machine & Build, from SCRATCH, the Head
   (a) svn co repository head
   (b) make all/runtests
   (c) If build fails
   - Evaluate Problem:
     i. If a quick-fix then FIX-IT in your own local sandbox
     ii. Start Over at 1),
     iii. Else if NOT a quick-fix then UNDO Commit to SVN
     iv. Communicate your DONE
     v. Start Over at Step 1)

   (d) 100% NO Errors or Warnings!
6. COMMUNICATE YOUR DONE BY PUTTING Sockmonkey Back in His Place!!

The process is short and simple. If a Build failure occurs during these steps it was fixed. If the failure

[18] This process outline was posted on a large Post-it in the MCATK bullpen until sometime in 2017.

[19] A Ten-Minute Build means to build and execute the tests in ten minutes.

happens while doing Step 5, then the failure was evaluated and you would decide if you can fix it "quickly" (usually time-blocked) and if not then back off your commit from the mainline source repository. Note that this particular process implies that integration is being done in a synchronous way and not an asynchronous way. In other words, a *mini-code freeze* until the developer with the commit token (Chuckhead the Sockmonkey) has rung the bell and returned the token to it place. Being a small team in a bullpen for co-location, and using "The One Button" and "Ten-Minute Builds" CI practices help make this process doable.

Bibliography

Adams, B., McIntosh, S., 2016. "Modern Release Engineering in a Nutshell – Why Researchers Should Care". In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). Vol. 5. pp. 78–90.

Adams, T., Nolen, S., Sweezy, J., Zukaitis, A., Campbell, J., Goorley, T., Greene, S., Aulwes, R., 2015. "Monte Carlo Application ToolKit (MCATK)". Annals of Nuclear Energy 82 (0), 41–47.

Adams, T. R., 2019. "Continuous Integration!". Tech. Rep. LA-UR-19-21074, Los Alamos National Labortary, Los Alamos, NM, USA, Presentation Slides.
URL https://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-19-21074

Amazon, 2019. "What is Continuous Integration?". https://aws.amazon.com/devops/continuous-integration, retrieved 11 January 2019.

Bahrs, P., No date: 2018. "Getting Started with Continuous Integration". https://www.ibm.com/cloud/garage/content/code/practice_continuous_integration, retrieved 2 November 2018.

Basu, S., 2017. "Continuous Integration: Its History and Benefits". https://dzone.com/articles/continuous-integration-and-its-whereabouts, retrieved 8 August 2018.

Beck, K., 1998. "Extreme Programming: A Humanistic Discipline of Software Development". In: Astesiano, E. (Ed.), Proceedings of International Conference on Fundamental Approaches to Software Engineering. Vol. 1382 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Germany, pp. 1–6.

Beck, K., 1999. "Embracing Change with Extreme Programming". Computer 32 (10), 70–77.
URL https://ieeexplore.ieee.org/abstract/document/796139

Beck, K., 2000. Extreme Programming Explained: Embrace Change. Addison-Wesley, Boston, MA, US.

Beck, K., Andres, C., 2004. Extreme Programming Explained: Embrace Change, 2nd Edition. Addison-Wesley, Boston, MA, US.

Bernardo, J. a. H., da Costa, D. A., Kulesza, U., 2018. "Studying the Impact of Adopting Continuous Integration on the Delivery Time of Pull Requests". In: Proceedings of the 15th International Conference on Mining Software Repositories. MSR '18. ACM, New York, NY, USA, pp. 131–141.

Bobrovskis, S., Jurenoks, A., 2018. "A Survey of Continuous Integration, Continuous Delivery and Continuous Deployment". In: Zdravkovic, J., Grabis, J., Nurcan, S., Stirna, J. (Eds.), Joint Proceedings of the BIR 2018 Short Papers, Workshops and Doctoral Consortium colocated with 17th International Conference Perspectives in Business Infomatics Research (BIR 2018) Stockholm, Sweden, September 24-26, 2018. Vol. 2218. pp. 314–322.
URL http://ceur-ws.org/Vol-2218/paper31.pdf

Bogard, J., 2017. "Trunk-Based Development or Pull Requests - Why Not Both?". https://jimmybogard.com/trunk-based-development-or-pull-requests-why-not-both/, retrieved 31 December 2018.

Booch, G., 1991. Object-Oriented Design: With Applications. The Benjamin/Cummings Publishing Company, Inc., Redwood, CA, US.

Booch, G., 1994. Object-Oriented Design and Analysis with Applications (2nd Edition). The Benjamin/Cummings Publishing Company, Inc., Redwood, CA, US.

Booch, G., 1996. Object Solutions: Managing the Object-Oriented Project. Pearson Education, Menlo Park, CA, US.

Bowyer, J., Hughes, J., 2006. "Assessing Undergraduate Experience of Continuous Integration and Test-Driven Development". In: Proceedings of the 28th International Conference on Software Engineering. ACM, pp. 691–694.

Brady, D., 2017. "Continuous Integration, not Continuous Isolation". https://damianbrady.com.au/2017/07/12/continuous-integration-not-continuous-isolation/, retrieved 8 October 2018.

Brandtner, M., Giger, E., Gall, H., Feb 2014. "Supporting Continuous Integration by Mashing-up Software Quality Information". In: 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE). pp. 184–193.

Brito, G., Terra, R., Valente, M. T., 2018. "Monorepos: A Multivocal Literature Review". arXiv preprint arXiv:1810.09477.
URL https://arxiv.org/pdf/1810.09477.pdf

Bugayenko, V., 2014a. "Continuous Integration Doesn't Work". https://devops.com/continuous-integration-doesnt-work, retrieved 30 November 2017.

Bugayenko, V., 2014b. "Continuous Integration Is Dead". http://www.yegor256.com/2014/10/08/continuous-integration-is-dead.html, retrieved 11 December 2017.

Campbell, R., 2012. "Software Development Practices: A Conversation with Steve McConnell". https://www.itprotoday.com/microsoft-visual-studio/software-development-practices-conversation-steve-mcconnell, retrieved 1 September

2018.

Cockburn, A., 2005. Crystal Clear: A Human-Powered Methodology for Small Teams. Addison-Wesley, Boston, MA, US.

Coelho, J., Valente, M. T., 2017. "Why Modern Open Source Projects Fail". In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ESEC/FSE 2017. ACM, New York, NY, USA, pp. 186–196.
URL http://doi.acm.org/10.1145/3106237.3106246

Cunningham, W., Contributors, Last Edited 2012. "WikiWikiWeb: Integration Hell". http://wiki.c2.com/?IntegrationHell, retrieved, 01 January 2018;.

Cunningham, W., Contributors, Last Edited 2014. "WikiWikiWeb: Continuous Integration". http://wiki.c2.com/?ContinuousIntegration, retrieved, 01 January 2018.

Cusumano, M. A., Selby, R. W., 1995. Microsoft Secrets. The Free Press, New York, NY, US.

Dabbish, L., Stuart, C., Tsay, J., Herbsleb, J., 2012. "Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository". In: Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work. CSCW '12. ACM, New York, NY, USA, pp. 1277–1286.
URL http://doi.acm.org/10.1145/2145204.2145396

Debbiche, A., Dienér, M., 2014. Assessing Challenges of Continuous Integration in the Context of Software Requirements Breakdown: A Case Study. Master's thesis, Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden.

Debbiche, A., Dienér, M., Berntsson Svensson, R., 2014. "Challenges When Adopting Continuous Integration: A Case Study". In: Jedlitschka, A., Kuvaja, P., Kuhrmann, M., Männistö, T., Münch, J., Raatikainen, M. (Eds.), Product-Focused Software Process Improvement. Springer International Publishing, Cham, pp. 17–32.

Deshpande, A., Riehle, D., 2008. "Continuous Integration in Open Source Software Development". In: Russo, B., Damiani, E., Hissam, S., Lundell, B., Succi, G. (Eds.), Open Source Development, Communities and Quality. Springer US, Boston, MA, pp. 273–280.

DORA, 2017. "Capital One Drives Continuous Delivery Improvement with Insights from DORA: A Case Study". https://devops-research.com/assets/capital_one_case_study.pdf, retrieved, 2018.

Driessen, V., 2010. "A Successful Git Branching Model". https://nvie.com/posts/a-successful-git-branching-model/, retrieved, 6 November 2018.

Duvall, P., 2006. "Automation for the People: Continuous Feedback: Receive Immediate Feedback with Every Source Code Change". https://www.ibm.com/developerworks/library/j-ap11146/j-ap11146-pdf.pdf, retrieved, 08 August 2018.

Duvall, P., 2007. "Automation for the People: Continuous Integration Anti-Patterns: Make Your Life with CI Easier by Learning What Not To Do". https://www.ibm.com/developerworks/library/j-ap11297/index.html, retrieved, 1 December 2017.

Duvall, P., 2008. "Automation for the People: Continuous Integration Anti-Patterns, Part 2: Make Your Life with CI Easier by Learning What Not To Do". https://www.ibm.com/developerworks/java/library/j-ap03048/j-ap03048-pdf.pdf, retrieved, 1 December 2017.

Duvall, P. M., Glover, A., Matyas, S., 2007. Continuous Integration: Improving Software Quality and Reducing Risk. Addison-Wesley, Upper Saddle River, NJ, US.

Ecker, R., 2016a. "Code Reviews Trunk Based Development". https://team-coder.com/code-reviews-in-trunk-based-development/, retrieved, 01 October 2018.

Ecker, R., 2016b. "From Git Flow to Trunk Based Development". https://team-coder.com/from-git-flow-to-trunk-based-development/, retrieved, 01 October 2018.

Farley, D., 2011. "Don't Feature Branch". http://www.davefarley.net/?p=160, retrieved, 15 October 2018.

Farley, D., April 2018a. "A Few Thoughts on Feature Flags". http://www.davefarley.net/?p=255, retrieved, 15 October 2018.

Farley, D., September 2018b. "CI and the Change Log". http://www.davefarley.net/?p=263, retrieved, 15 October 2018.

Farley, D., March 2018c. "Continuous Integration and Feature Branching". http://www.davefarley.net/?p=247, retrieved, 15 October 2018.

Feitelson, D. G., Frachtenberg, E., Beck, K. L., July 2013. "Development and Deployment at Facebook". IEEE Internet Computing 17 (4), 8–17.

Fernandez, S., 2018. "Back to the Roots: Towards True Continuous Integration (Part I)". https://dzone.com/articles/back-to-the-roots-towards-true-continuous-integrat, retrieved, 08 August 2018.

Fitzgerald, B., Stol, K.-J., 2017. "Continuous Software Engineering: A Roadmap and Agenda". Journal of Systems and Software 123, 176–189.
URL http://www.sciencedirect.com/science/article/pii/S0164121215001430

Forsgren, N., Brown, A., Humble, J., Kersten, N., Kim, G., 2016. "2016 State of DevOps Report". Tech. rep., Puppet and DevOps Research and Assessment.
URL http://nicolefv.com/s/2016-State-of-DevOps-Report.pdf

Forsgren, N., Humble, J., Kim, G., 2018a. "Accelerate: State of DevOps 2018: Strategies for a New Economy Report". Tech. rep., DevOps Research and Assessment.
URL https://cloudplatformonline.com/2018-state-of-devops.html

Forsgren, N., Humble, J., Kim, G., 2018b. Accelerate: The Science of Lean Software and DevOps Building and Scaling High Performing Technology Organizations, 1st Edition. IT Revolution Press.

Forsgren, N., Kim, G., Kersten, N., Humble, J., 2014. "2014

State of DevOps Report". Tech. rep., Puppet, IT Revolution Press, and ThoughtWorks.
URL http://nicolefv.com/s/2014-state-of-devops-report-aezm.pdf

Forsgren, N., Kim, G., Kersten, N., Humble, J., 2015. "2015 State of DevOps Report". Tech. rep., Puppet and IT Revolution Press.
URL http://nicolefv.com/s/2015-DevOps-Report-final-with.pdf

Forsgren, N., Kim, G., Kersten, N., Humble, J., Brown, A., 2017. "2017 State of DevOps Report". Tech. rep., DevOps Research and Assessment.
URL http://nicolefv.com/s/2017SODOR-FINAL.pdf

Fowler, M., 2006a. "BranchByAbstraction". https://martinfowler.com/bliki/BranchByAbstraction.html, retrieved, 21 October 2018.

Fowler, M., 2006b. "SemanticDiffusion". https://martinfowler.com/bliki/SemanticDiffusion.html, retrieved, 27 August 2018.

Fowler, M., 2009. "FeatureBranch". https://martinfowler.com/bliki/FeatureBranch.html, retrieved, 06 October 2018.

Fowler, M., 2010. "FeatureToggle". https://martinfowler.com/bliki/FeatureToggle.html, retrieved, 21 October 2018.

Fowler, M., 2011. "FrequencyReducesDifficulty". https://martinfowler.com/bliki/FrequencyReducesDifficulty.html, retrieved, 22 November 2018.

Fowler, M., 2017. "ContinuousIntegrationCertification". https://martinfowler.com/bliki/ContinuousIntegrationCertification.html, retrieved, 08 October 2018.

Fowler, M., 2018. "My Foreword to Accelerate". https://martinfowler.com/articles/accelerate-foreword.html, retrieved, 08 October 2018.

Fowler, M., Foemmel, M., 2000. "Continuous Integration (Original Version)". https://www.martinfowler.com/articles/originalContinuousIntegration.html, retrieved, 01 January 2018.

Fowler, M., Foemmel, M., 2006. "Continuous Integration". https://www.martinfowler.com/articles/continuousIntegration.html, retrieved, 01 January 2018.

Fuggetta, A., Di Nitto, E., 2014. "Software Process". In: Proceedings of the on Future of Software Engineering. FOSE 2014. ACM, New York, NY, USA, pp. 1–12.
URL http://doi.acm.org/10.1145/2593882.2593883

Gadzinowski, K., 2018. "Trunk-Based Development vs. Git Flow". https://www.toptal.com/software/trunk-based-development-git-flow, retrieved, 06 October 2018.

Garnichaud, N., 2012. "What Exactly is DevOps?". Dr. Dobb's: The World of Software Development.
URL http://www.drdobbs.com/architecture-and-design/what-exactly-is-devops/240009147#

Gary, K., Enquobahrie, A., Ibanez, L., Cheng, P., Yaniv, Z., Cleary, K., Kokoori, S., Muffih, B., Heidenreich, J., 2011. "Agile Methods for Open Source Safety-critical Software". Software: Practice and Experience 41 (9), 945–962.

Gaston, D. R., Peterson, J. W., Permann, C. J., Andrs, D., Slaughter, A. E., Miller, J. M., 2015. "Continuous Integration for Concurrent Computational Framework and Application Development". Journal Open Research Software 2 (1), p.e10.
URL http://doi.org/10.5334/jors.as

Gilb, T., 1981. "Evolutionary Development". SIGSOFT Software Engineering Notes 6 (2), 17.
URL http://doi.acm.org/10.1145/1010865.1010868

Gilb, T., 1988. Principles of Software Engineering Management. Pearson Education Limited, Harlow Essex, Great Britian.

GitHub, 2017. "Understanding the GitHub Flow". https://guides.github.com/introduction/flow/, retrieved, 06 October 2018.

Glover, A., 2007. "Spot Defects Early With Continuous Integration". https://www.ibm.com/developerworks/java/tutorials/j-cq11207/j-cq11207-pdf.pdf, retrieved, 22 December 2017.

Gousios, G., Pinzger, M., Deursen, A. v., 2014. "An Exploratory Study of the Pull-based Software Development Model". In: Proceedings of the 36th International Conference on Software Engineering. ICSE 2014. ACM, New York, NY, USA, pp. 345–355.
URL http://doi.acm.org/10.1145/2568225.2568260

Gousios, G., Storey, M., Bacchelli, A., May 2016. "Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective". In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). pp. 285–296.

Gousios, G., Zaidman, A., Storey, M.-A., van Deursen, A., 2015. "Work Practices and Challenges in Pull-based Development: The Integrator's Perspective". In: Proceedings of the 37th International Conference on Software Engineering - Volume 1. ICSE '15. IEEE Press, Piscataway, NJ, USA, pp. 358–368.
URL http://dl.acm.org/citation.cfm?id=2818754.2818800

Grebenyuk, A., 2018. "Trunk-Based Development". http://kean.github.io/post/trunk-based-development, retrieved, 06 October 2018.

Hamdan, S., Alramouni, S., 2015. "A Quality Framework for Software Continuous Integration". Procedia Manufacturing 3, 2019–2025, 6th International Conference on Applied Human Factors and Ergonomics (AHFE 2015) and the Affiliated Conferences, AHFE 2015.

Hammant, P., 2013. "What is Truck-Based Development?". https://paulhammant.com/2013/04/05/what-is-trunk-based-development/, retrieved, 06 October 2018.

Hammant, P., Smith, S., 2018. Trunk Based Development - Webpage. https://trunkbaseddevelopment.com/, retrieved, 06 October 2018.

Henney, K., 2018. “GOTO Chicago 2018: Old is the New New”. https://www.youtube.com/watch?v=AbgsfeGvg3E, retrieved,2 November 2018.

Hilton, M., Nelson, N., Dig, D., Tunnell, T., Marinov, D., et al., 2016a. “Continuous Integration (CI) Needs and Wishes for Developers of Proprietary Code”. Tech. rep., Corvallis, OR: Oregon State University, Dept. of Computer Science.
URL http://dspace-ir.library.oregonstate.edu/xmlui/handle/1957/59856

Hilton, M., Tunnell, T., Huang, K., Marinov, D., Dig, D., 2016b. “Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects”. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. ACM, pp. 426–437.

Hodgson, P., 2010. “Feature Toggles (aka Feature Flags)”. https://martinfowler.com/articles/feature-toggles.html, retrieved, 21 October 2018.

Holck, J., Jørgensen, N., 2003. “Continuous Integration and Quality Assurance: A Case Study of Two Open Source Projects”. Australasian Journal of Information Systems 11 (1).

Humble, J., 2018. “ContinuousDelivery.com Webpage”. https://continuousdelivery.com/, retrieved, 2018.

Humble, J., Farley, D., 2010. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, Upper Saddle River, NJ, US.

Husted, T., 2014. “Why Feature Branches are a Good Thing”. https://www.nimbleuser.com/blog/why-feature-branches-are-a-good-thing, retrieved, 06 October 2018.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., Damian, D., 2014. “The Promises and Perils of Mining GitHub”. In: Proceedings of the 11th Working Conference on Mining Software Repositories. MSR 2014. ACM, New York, NY, USA, pp. 92–101.
URL http://doi.acm.org/10.1145/2597073.2597074

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., Damian, D., Oct 2016. “An In-depth Study of the Promises and Perils of Mining GitHub”. Empirical Software Engineering 21 (5), 2035–2071.
URL https://doi.org/10.1007/s10664-015-9393-5

Karanth, D., 2017a. “Continuous Integration Part 1: The Fundamentals”. http://techtowntraining.com/resources/blog/continuous-integration-part-1-fundamentals, retrieved, 10 August 2017.

Karanth, D., 2017b. “Continuous Integration Part 2: CI Server & Toolkit”. http://techtowntraining.com/resources/blog/continuous-integration-part-2-ci-server-toolkit, retrieved, 10 August 2017.

Karanth, D., 2017c. “Continuous Integration Part 3: Best Practices”. http://techtowntraining.com/resources/blog/continuous-integration-part-3-best-practices, retrieved, 10 August 2017.

Kim, S., Park, S., Yun, J., Lee, Y., 2008. “Automated Continuous Integration of Component-Based Software: An Industrial Experience”. In: Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society, pp. 423–426.

Lacoste, F. J., Aug 2009. “Killing the Gatekeeper: Introducing a Continuous Integration System”. In: 2009 Agile Conference. pp. 387–392.

Lamourine, M., 2018. “Book Review of Accelerate: Building and Scaling High Performing Technology Organizations”. ;login: The Usenix Magazine 43 (3), 60–61.
URL https://www.usenix.org/system/files/login/articles/login_fall18_15_books.pdf

Larman, C., 2004. Agile and Iterative Development. Addison-Wesley, Boston, MA, US.

Legay, D., Decan, A., Mens, T., 2018. “On the Impact of Pull Request Decisions on Future Contributions”. arXiv preprint arXiv:1812.06269.
URL https://arxiv.org/pdf/1812.06269.pdf

Limoncelli, T., 2014. “Yes, You Really Can Work From HEAD”. http://everythingsysadmin.com/2014/03/yes-you-really-can-work-from-head.html, retrieved, 06 October 2018.

Mahlberg, M., 2014. “There is No Such Thing as a Continuous Integration Server”. http://agile-aspects.michaelmahlberg.com/2014/12/there-is-no-such-thing-as-continuos.html, retrieved, 8 January 2018.

Martensson, T., Hammarström, P., Bosch, J., Aug 2017. “Continuous Integration is Not About Build Systems”. In: 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA). pp. 1–9.

McConnell, S., 1996a. “Daily Build and Smoke Test”. IEEE Software July, 144.

McConnell, S., 1996b. Rapid Development. Microsoft Press, Redmond, WA, US.

McConnell, S., 2004. Code Complete, 2nd Edition. Microsoft Press, Redmond, WA, US.

McConnell, S., 2008. “Agile Software: Business Impact and Business Benefits”. http://www.construx.com/10x_Software_Development/Agile_Software_Business_Impact_and_Business_Benefits, retrieved, 6 November 2017.

Mergel, I., 2015. “Open Collaboration in the Public Sector: The Case of Social Coding on GitHub”. Government Information Quarterly 32 (4), 464 – 472.
URL http://www.sciencedirect.com/science/article/pii/S0740624X15300095

Meyer, B., 2014a. Agile! The Good, Bad, and Ugly. Springer International Publishing, Switzerland.

Meyer, M., 2014b. “Continuous Integration and Its Tools”. IEEE Software 31 (3), p. 14.

Miller, A., Aug 2008. “A Hundred Days of Continuous Integration”. In: Agile 2008 Conference. pp. 289–293.

Mówiński, K., 2018. “Escape from Merge Hell: Why I Prefer Trunk-Based Development Over Feature Branching and GitFlow”. https://stxnext.com/blog/2018/02/28/escape-merge-hell-why-i-prefer-trunk-

based-development-over-feature-branching-and-gitflow/, retrieved, 6 October 2018.

Newman, S., 2017. "GOTO Chicago 2017: Feature Branches and Toggles in a Post-GitHub World". https://www.youtube.com/watch?v=lqRQYEHAtpk, retrieved, 15 October 2018.

North, D., 2006. "Continuous Build is Not Continuous Integration". https://dannorth.net/2006/03/22/continuous-build-is-not-continuous-integration/, retrieved, 15 December 2018.

Ortu, M., Pinna, A., Tonelli, R., Marchesi, M., Bowes, D., Destefanis, G., 2018. "Angry-Builds: An Empirical Study Of Affect Metrics and Builds Success on GitHub Ecosystem". In: XP '18 Companion: XP '18 Companion, May 21–25, 2018, Porto, Portugal. ACM, New York, NY, USA.
URL https://www.researchgate.net/publication/328069671_Angry-Builds_An_Empirical_Study_Of_Affect_Metrics_and_Builds_Success_on_GitHub_Ecosystem

Phillips, S., Sillito, J., Walker, R., 2011. "Branching and Merging: An Investigation into Current Version Control Practices". In: Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering. CHASE '11. ACM, New York, NY, USA, pp. 9–15.
URL http://doi.acm.org/10.1145/1984642.1984645

Pignatelli, N., 2018. "How to Get Continuous Integration Right. The Secret to Deliver at Full Throttle is Not About Your Tools". https://hackernoon.com/how-to-get-continuous-integration-right-77bda4bc0d1f, retrieved, 28 August 2018.

Pilone, D., Miles, R., 2008. Head First Software Development. O'Reilly Media Inc., Sebastopol, CA, US.

Pinto, G., Castor, F., Bonifacio, R., Rebouças, M., 2018. "Work Practices and Challenges in Continuous Integration: A Survey with Travis CI Users". Software: Practice and Experience 48 (12), 2223–2236.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2637

Pittet, S., 2018. "How To Get Started with Continuous Integration". https://www.atlassian.com/continuous-delivery/how-to-get-to-continuous-integration, retrieved 28 August 2018.

Poppendieck, M., Poppendieck, T., 2010. Leading Lean Software Development: Results Are Not the Point. Addison-Wesley, Boston, MA, US.

Potvin, R., Levenberg, J., July 2016. "Why Google Stores Billions of Lines of Code in a Single Repository". Communications of the ACM 59 (7), 78–87.

Prentice, A., 2009. "Testing: You Don't Have to be Agile to be Agile". https://www.atlassian.com/blog/archives/testing_you_dont_have_to_be_agile_to_be_agile, retrieved, 8 August 2018.

Prince, S., 2017. "It's Not CI, it's Just CI Theatre". https://www.gocd.org/2017/05/16/its-not-CI-its-CI-theatre.html, retrieved 11 January 2018.

Radigan, D., 2018. "Continuous Integration, Explained". https://www.atlassian.com/continuous-delivery/continuous-integration-intro, retrieved, 28 August 2018.

Rahman, A., Agrawal, A., Krishna, R., Sobran, 2018. "Characterizing The Influence of Continuous Integration". ACM.
URL https://akondrahman.github.io/papers/swan2018_ci.pdf

Rahman, A., Agrawal, A., Krishna, R., Sobran, A., Menzies, T., 2017. "Continuous Integration: The Silver Bullet?". arXiv preprint arXiv:1711.03933.

Rahman, M. T., Querel, L.-P., Rigby, P. C., Adams, B., 2016. "Feature Toggles: Practitioner Practices and a Case Study". In: Proceedings of the 13th International Conference on Mining Software Repositories. MSR '16. ACM, New York, NY, USA, pp. 201–211.
URL http://doi.acm.org/10.1145/2901739.2901745

Rebouças, M., Santos, R. O., Pinto, G., Castor, F., 2017. "How Does Contributors' Involvement Influence the Build Status of an Open-source Software Project?". In: Proceedings of the 14th International Conference on Mining Software Repositories. MSR '17. IEEE Press, Piscataway, NJ, USA, pp. 475–478.
URL https://doi.org/10.1109/MSR.2017.32

Rogers, R. O., 2004. "Scaling Continuous Integration". In: International Conference on Extreme Programming and Agile Processes in Software Engineering. Springer, pp. 68–76.

Ropa, S., 2018. "A Craftsman Looks at Continuous Integration". https://about.gitlab.com/2018/01/17/craftsman-looks-at-continuous-integration, retrieved, 28 August 2018.

Rossi, C., Shibley, E., Su, S., Beck, K., Savor, T., Stumm, M., 2016. "Continuous Deployment of Mobile Software at Facebook (Showcase)". In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. FSE 2016. ACM, New York, NY, USA, pp. 12–23.
URL http://doi.acm.org/10.1145/2950290.2994157

Savor, T., Douglas, M., Gentili, M., Williams, L., Beck, K., Stumm, M., May 2016. "Continuous Deployment at Facebook and OANDA". In: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C). pp. 21–30.

Setter, M., 2017a. "Reduce Production Bugs with Continuous Integration". https://blog.codeship.com/reduce-production-bugs-with-continuous-integration, retrieved 28 September 2018.

Setter, M., 2017b. "Why Continuous Integration Is Important". https://blog.codeship.com/continuous-integration-important, retrieved 24 October 2017.

Shahin, M., Babar, M. A., Zhu, L., 2017. "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices ". IEEE Access 5, 3909–3943.

Shore, J., 2005a. "Continuous Integration is an Attitude, Not a Tool". http://www.jamesshore.com/Blog/Continuous-Integration-is-an-Attitude.html, retrieved, 6 November 2017.

Shore, J., 2005b. "Why I Don't Like Cruise Control". http://www.jamesshore.com/Blog/WhyIDontLikeCruiseControl.html, retrieved, 01 December 2017.

Shore, J., 2006. "Continuous Integraton on a Dollar a Day". http://www.jamesshore.com/Blog/Continuous-Integration-on-a-Dollar-a-Day.html, retrieved, 6 November 2017.

Shore, J., 2008. "Forces Affecting Continuous Integration". http://www.jamesshore.com/Blog/Forces-Affecting-Continuous-Integration.html, retrieved, 6 November 2017.

Shore, J., 2012. "Continuous Integration with Git". https://www.letscodejavascript.com/v3/transcripts/lessons_learned/1, retrieved, 6 November 2017.

Shore, J., Warden, S., 2008. The Art of Agile Development. O'Reilly Media, Inc., Sebastopol, US.

Slaughter, A. E., Peterson, J. W., Gaston, D. R., Permann, C. J., Andrs, D., Miller, J. M., 2015. "Continuous Integration for Concurrent MOOSE Framework and Application Development on GitHub". Journal of Open Research Software 3 (INL/JOU-15-34179).

Sletholt, M. T., Hannay, J., Pfahl, D., Benestad, H. C., Langtangen, H. P., 2011. "A Literature Review of Agile Practices and Their Effects in Scientific Software Development". In: Proceedings of the 4th International Workshop on Software Engineering for Computational Science and Engineering. SECSE '11. ACM, New York, NY, USA, pp. 1–9.

Staron, M., Meding, W., Soder, O., Back, M., 2018. "Measurement and Impact Factors of Speed of Reviews and Integration in Continuous Software Engineering". Foundations of Computing and Decision Sciences 43 (4), 281–303.

StrideNews, 2017. "The Rabbit Hole: 30. Truck Based Development vs Gitflow". https://www.stridenyc.com/podcasts/30-trunk-based-development-vs-gitflow, retrieved 30 December 2018.

Ståhl, D., Bosch, J., 2014. "Modeling Continuous Integration Practice Differences in Industry Software Development". The Journal of Systems and Software 87 (1), 48–59.

Ståhl, D., Mårtensson, T., Bosch, J., 2017. "The Continuity of Continuous Integration: Correlations and Consequences". Journal of Systems and Software 127, 150–167.

Tørresdal, J. A., 2016a. "Continuous Integration (CI) – Is That What We're Doing?". https://mrdevops.io/continuous-integration-ci-is-that-what-were-doing-e3137dfd39fd, retrieved, 08 August 2018.

Tørresdal, J. A., 2016b. "Trunk-Based Development". https://mrdevops.io/trunk-based-development-8376fe577c11, retrieved, 08 August 2018.

Tørresdal, J. A., 2017. "If You Still Insist on Feature Branching, You are Hurting Your Business and Our Profession". https://mrdevops.io/if-you-still-insist-on-feature-branching-you-are-hurting-your-business-and-our-profession-32e1109d4594, retrieved, 08 August 2018.

Trahan, T. J., Adams, T. R., Burke, T. P., Dixon, D. A., McCartney, A. P., Nolen, S. D., Sweezy, J. E., Werner, C., 2018. "Monte Carlo Application ToolKit (MCATK), MonteRay and Talon GPU Tally Libraries". Tech. Rep. LA-UR-18-27957, Los Alamos National Labortary, Los Alamos, NM, USA, 20th Topical Meeting of the Radiation Protection & Shielding Division of ANS ; 2018-08-26 - 2018-08-31 ; Santa Fe, New Mexico, United States.
URL https://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-18-27957

Van Der Storm, T., 2008. "Backtracking Incremental Continuous Integration". In: Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on. IEEE, pp. 233–242.

Vasilescu, B., van Schuylenburg, S., Wulms, J., Serebrenik, A., van den Brand, M. G. J., Sept 2014. "Continuous Integration in a Social-Coding World: Empirical Evidence from GitHub". In: 2014 IEEE International Conference on Software Maintenance and Evolution. pp. 401–405.

Vasilescu, B., Yu, Y., Wang, H., Devanbu, P., Filkov, V., 2015. "Quality and Productivity Outcomes Relating to Continuous Integration in GitHub". In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ESEC/FSE 2015. ACM, New York, NY, USA, pp. 805–816.

Warden, S., 2003. Extreme Programming Pocket Guide. O'Reilly Media, Incorporated.

Wikipedia, 2017. "Continuous Integration". https://en.wikipedia.org/wiki/Continuous_integration, retrieved 15 December 2017.

Wikipedia, 2018. "Anti-pattern". https://en.wikipedia.org/wiki/Anti-pattern, retrieved 26 December 2018.

Yu, Y., Wang, H., Filkov, V., Devanbu, P., Vasilescu, B., May 2015. "Wait for It: Determinants of Pull Request Evaluation Latency on GitHub". In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. IEEE, pp. 367–371.

Zhao, Y., Serebrenik, A., Zhou, Y., Filkov, V., Vasilescu, B., 2017. "The Impact of Continuous Integration on Other Software Development Practices: A Large-scale Empirical Study". In: Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering. ASE 2017. IEEE Press, Piscataway, NJ, USA, pp. 60–71.
URL http://dl.acm.org/citation.cfm?id=3155562.3155575

Zilberfeld, G., 2017. "You're Doing It All Wrong: Continuous Integration". http://www.gilzilberfeld.com/2017/05/youre-doing-it-wrong-continuous-integration.html, retrieved, 06 November 2017.

About The Author

Terry Adams has close to 35 years of experience in computational physics that covers performing computational simulations, supporting user applications, developing new scientific algorithms, and doing scientific software development. His first computational physics simulation was modelling the calibration of a small proton accelerator using Lotus 123. This lead to computational quantum chemistry simulations that modelled muon spin research for solid-state physics experiments at LAMPF/LANL. Terry worked on the commercial quantum chemistry software package for Q-Chem, Inc. for nearly four years. There he learned a good bit of *what not to do* in software development and found the need for, what he now knows as, software engineering practices and tools. Starting in 1998, he worked for Silicon Graphics, Inc. and then LANL where he gained parallel high-performance computing (HPC) experience working with, and on, several ASCI projects. Terry started practicing continuous integration (CI) in 2008 at the start of the Monte Carlo Application ToolKit (MCATK) effort.